DOCUMENT RESUME

ED 287 442 IR 012 825

AUTHOR Seidman, Robert H.
TITLE Research on Teaching and Learning Computer Programming Symposium.
PUB DATE Apr 87
NOTE 15p.; Paper presented to Symposium (15.03) at the Annual Meeting of the American Educational Research Association (Washington, DC, April 20-24, 1987).
PUB TYPE Viewpoints (120) -- Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Cognitive Processes; *Computer Science Education; Group Instruction; *Instructional Design; Metacognition; Models; *Programing; *Programing Languages; Psychological Studies
IDENTIFIERS BASIC Programing Language; LOGO Programing Language

ABSTRACT

Four conference papers are reviewed in this introduction to a symposium. The first paper is by Clements and Merriman, who make a case for the reflection of Steinberg's information processing componential model of cognitive processes in the LOGO language and computational environment; present a series of experiments that attempted to tailor the environment to aspects of the theoretical model and to assess transfer of componential and metacomponential skills; and speculate on how to structure an ideal LOGO environment to facilitate transfer of cognitive skills. In the second paper, Perkins, Schwartz, and Simmons report their findings from interviews with naive BASIC programmers, including their need for a mental model of the computing process, lack of good problem solving strategies, and problems in personal confidence and control. A metacourse designed to teach metacognitive skills is proposed to address these problems. Fay and Mayer, in the third paper, argue that LOGO mastery is affected by the cognitive misconceptions of naive learners, and provide a computational model of such misconceptions. In the final paper, Webb and Lewis confirm that group learning of programming has positive results and propose a metacourse to promote efficacious group behavior. (MES)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED287442

American Educational Research Association Annual Meeting

Washington, D.C. - April 20-24, 1987

Research on Teaching and Learning Computer Programming Symposium (15.03)

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

X This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

Robert H. Seidman - Chair/Discussant
New Hampshire College Graduate School
2500 N. River Road
Manchester, N.H. 03104

IR013825

BEST COPY AVAILABLE

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Robert H. Seidman

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

Welcome to the "Research on Teaching and Learning Computer Programming" Symposium. I will read the objectives written by the Symposium organizer, Richard Mayer.

"The purpose of this symposium is to investigate how students learn programming languages and how instruction can affect learning. In particular, the symposium focuses on two major issues: (1) What are the cognitive consequences of learning a programming language. A more specific version of this question is: When a student learns to use a programming language, does the student also learn a procedural way of thinking that can be applied to problems outside the domain of programming? (2) What kind of instruction enables students to productively learn programming? A more specific version of this question is: How much and what kind of guidance should be given to novices who are learning to program?"

"Programming languages are being incorporated into school curricula, often without the benefit of a research base. This symposium brings together some of the leading and most active researchers in the area of learning and teaching of programming languages. Symposium participants have been requested to present their research and its implications for educational practice. Two important research implications for education addressed and clarified in this symposium are: (1) that learning to program can transfer to some problem solving tasks but that such transfer is limited, and (2) that specific kinds of guidance are needed for many novices to learn programming."

I ask each participant to take no more than 10 minutes so that we can leave adequate time for questions. We will proceed in the order listed in the program.

I am pleased to inform you that more in-depth versions of the Conference papers will appear later this year in a volume edited by Richard Mayer, titled: Teaching and Learning Computer Programming: Multiple Research Perspectives. It will be published by Laurance Erlbaum Associates.

One way to discuss a set of conference papers such as these is to assess the importance of the research question addressed, determine whether the research has been carried out well, examine the appropriateness of the conclusions, and explore the important implications of the combined research endeavors.

Given the limited time here, my approach as discussant is to briefly synthesize the research, point out what I think are complementary as well as conflicting themes, and assess their contributions to and implications for theory, knowledge and practice. Finally, I mean to mention a new development not addressed in this symposium, and generally neglected in the mainstream educational computing research community.

My regret is that I received only four papers in sufficient time for review.

The Clements and Merriman paper sets the framework for discussing the others. It is divided into three parts. The first makes a case for the reflection of Steinberg's information processing componential model of cognitive processes in the LOGO language and computational culture (or environment). The second

section presents a series of experiments that try to tailor the environment to aspects of the theoretical model and to assess transfer of componential and metacomponential skills. The third part of the paper presents related research and speculates on how to structure an ideal Logo environment to facilitate transfer of cognitive skills.

This is an important paper for several reasons. It embeds Logo and its environment within a theortical framework which provides a standard against which to measure efficacy and it develops new instruments to measure transfer of specific cognitive skills. The paper provides many concrete suggestions for practitioners who want to tailor Logo enviornments for maximum effectivenes. Finally, the authors put forward the seemingly outlandish proposition that mastery of Logo programming may not, in and of itself, be a factor in the transfer of cognitive skills.

It shouldn't be surprising that the Logo language and environment are somewhat isomorphic to the cognitive theory proposed. After all, the theoretical model is itself derived from computer information processing theory and software implementation. The theory itself draws heavily on computational metaphors. Thus, I would expect almost any procedural computer language (and its accompanying environment) to reflect the theory in a more than superficial way. That Logo has a physical and screen "turtle" makes it also attractive to Piagetian developmentalists who see a potential "matching environment" between the concrete and formal operational stages. But I digress.

Clements and Merriman do an excelent job of pointing out specific correspondences between aspects of the theory and aspects of Logo and its environment. They also, in a way, take the pressure off the Logo language

itself by introducing what Perkin's calls, in his paper, a sort of "metacourse" which includes explicit instruction (through the "homunculi") to help students exercise their metacomponential and/or knowledge acquisition skills.

Clements and Merriman report mixed success for their experiment. As is the case in most pioneering research, they raise more questions than they answer. What can they tell us about the validity and reliability of their new assessment instruments? They readily admit that these instruments lacked sufficient items to allow accurate statistical discrimination of metacomponents. Are there other situations and instruments to measure transfer of effects such as these? How can they control for the individual cognitive predispositions of novice programmers as pointed out by Fay and Mayer? It would be interesting to see whether exposure to Logo and its accompanying environment affect skill transfer as defined by other cognitive theories.

But most importantly, Clements and Merriman did not care to measure mastery of Logo programming and then correlate it with cognitive skill transfer. This is an interesting approach. In their literature review they find that mastery of programming was not consistently a significant factor in transfer of cognitive skills. What then is a significant factor? Is it the overall environment within which Logo is embedded? Clement and Merriman seem to think so with respect to their environment. The fact that cognitive skill transfer can apparently take place without mastering Logo programming could serve as an argument for not teaching Logo but keeping aspects of its environment (the metacourse). Although, it might be shown that the vehicle, Logo, is indispensible for making the environment work and therefore is indeed a necessary component. This is food for thought and future research.

While Clements and Merriman believe that environemnt, rather than programming mastery, might be more decisive in aiding cognitive skill transfer, Perkins, Schwartz and Simmons argue that enviornment is crucial in making it easier to learn programming. Unlike Clements and Merriman, their environment was not designed specifically to promote transfer. Both research groups do believe that teaching a computer programming language is not enough. A special environment has to be created. This, of course, was Papert's original point although he apparently claimed that Logo alone provided such an environment. Now we are seeing a separate environment being fleshed out within an overall theoretical paradigm.

Perkins, Schwartz and Simmons quite aptly point out that procedural programming is problem-solving intensive and precision intensive and therefore hard to learn. (This supports the argument for abandoning procedural programming in favor of logic programming. More later.) From the results of clinical interviews of naive BASIC programmers they discover: the need for an efficacious mental model of the computing machine or process; the lack of good problem solving strategies; and problems in personal confidence and control. All influence BASIC mastery.

The solution proposed was to construct a "metacourse," which is a series of lessons put together to address these problems. The course is designed to teach metacognitive skills (such as those presented by Clements and Merriman) and is as Perkins, Schwartz and Simmons point out, an experiment in instructional design.

This research, unfinished as it is, represents a significant attempt to

construct an appropriate environment. However, it is not clear from the description of the "paper computer" and "interpreter" just what kind of mental computer model the students are presented with. Are students led to an understanding of the functional components of the computer and to its fetch/execute cycle? Is it important to know this? How does the "interpreter" illustrate the logic underlying the IF-THEN conditional? There is evidence that indicates that this logical conditional is naturally interpreted by children in a biconditional manner which may be inconsistent with the standard computational interpretation (Seidman, 1981, 1986). How are the notions of patterns and metapatterns presented? What is the explicit theory behind the course design and pedagogy? How important is method of presentation to student understanding?

The notion of metacourse is a refreshing one since it appears to raise the teaching of programming from the merely technical to something that potentially encompasses the education of higher level skills. In my view, it is important to see more curriculum details in order to better understand the study's results.

The study's results were admittedly confounded by the inability to factor out the higher ability of the control group by analysis of covariance. Dropping high ability subjects is less acceptable. Teacher training problems may also have clouded the results. Also, the metacourse was not explicitly designed to promote transfer.

Perkins, Schwartz and Simmons wisely make the point that the metacourse may be able to be oriented toward one or the other but not both: (1) good programming and (2) transfer of general cognitive skills. In fact, this particular

metacourse was focused specifically on good programming. The idea of a "bridge course" specifically addresses the transfer of cognitive skills.

In the end, Perkins, Schwartz and Simmons argue convincingly for programming instruction (within an appropriate environment) toward three distinct ends: good programming (technical), cognitive skills (higher level competencies) and cultural (societal demands/understanding). They support the notion that computer programming can be a vehicle in the latter two areas.

I note with interest that Perkins, Schwartz, Simmons and Clements and Merriman all agree that mere exposure to a programming language is not enough (for either mastery or transfer of cognitive skills) and that technical instruction must be augmented by direct or indirect instruction geared toward a higher level. Can we call this "metainstruction?"

The efforts of these two research teams in this area can have great future impact on the general and widespread teaching of computer programming in the nation's schools. This research needs to be carried on if we are to ever hope to educate students beyond the level of technical programming skills.

Fay and Mayer make an important contribution to our understanding and awareness of misconceptions that naive programmers bring to the learning experience. Their research complements the previous two papers in the sense that it shows that not only do we need appropriate environments surrounding the teaching of programming, but, we also need to be able to diagnose and appropriately match learners to those environments. Fay and Mayer's specific contribution is the identification of the egocentric conception of space and the undiscriminated conception of command with regard to the Logo graphical turtle environment. In addition, they present a computational model of the dimensions of these misconceptions.

Fay and Mayer are to be complimented on their conceptual clarity and very tightly controled experiment. However, I have a very specific concern about the terminology used in their Logo instructions. I wonder whether the word "turn" might have been misleading to the naive programmer. Our everyday experience with turning usually entails both rotating and moving. Whether riding a bike, walking or driving a car, to turn usually means to move and to rotate at the same time. Perhaps the word "rotate" or a phrase equivalent to "rotate about the midpoint of the turtle" might make the meaning of the turn command less confusing and thus affect the turn-and-move misinterpretation.

Fay and Mayer's admonition to offer programming at a developmentally appropriate level needs to be taken very seriously. I would think that readiness to learn a concept (or the need to unlearn a previously held one) might be crucial for learner success. But just what is the nature of the match between programming instruction and appropriate cognitive developmental level? Furthermore, on what particular dimensions can this match take place? Fay and

Mayer have identified one such dimension. There are surely others. I would offer the IF-THEN conditional as another candidate. It reflects the notion of deductive necessity which appears in different ways at different Piagetian developmental levels. This command is fundamental in any procedural computer language and is the nearest explicitly logical programming language construct.

Fay and Mayer would claim that Logo mastery is affected by learner cognitive misconceptions. Clements and Merriman might say that while this may be true it is not crucial to the transfer of cognitive skills. What's a practitioner to think of this much less do about it? Either mastering a programming language is important or it is not. Do we really need, as Fay and Mayer suggest, to intervene to correct misconceptions if indeed we don't really care about programming mastery? After all, Perkins, Schwartz and Simmons's "bridge course" is directed toward cognitive skills transfer using programming as a vehicle. Would we want students to be frustrated with not mastering the language but satisfied that they have mastered higher level cognitive skills?

The Webb and Lewis paper is an important contribution to the area of how to structure the social use of computers with regard to learning computer programming. They show that actively participating in group discussions, by giving and receiving explanations and input suggestions, is beneficial. Overall, Webb and Lewis confirm that group learning of programming seems to have positive results.

Webb and Lewis found that student discussions with each other about planning and debugging strategies were positively related to achievement. Most interesting to me was the result that student verbalization of planning and debugging to an instructor was negatively or not related to programming achievement. This calls into question the role of the instructor in the teaching/learning process. Implications for instructional design are obvious. I found Webb and Lewis's detective work in finding (from the research results) a plausible explanation of this phenomenon truly impressive. This is a fine example of educational research detective work.

Webb and Lewis were able to conclude that group initiating behavior was proabably not based on the group's need for help and that the instructor's behavior was mostly reactive and indirect. Thus, it was group working style, more than anything else, that accounted for the high frequency of asking the instructor questions. The help that the instructor did provide may have then been an impediment to students developing their own problem solving strategies.

Webb and Lewis propose a "metacourse" of sorts (perhaps I should call it a

"shadow course") that would augment the programming environment to promote the kind of group behavior deemed efficacious to taking full advantage of the computer programming learning environment. They suggest that the instructor, given the results of their study, should intervene as little as possible so that students would be forced to verbalize their problem solving strategies to each other.

It is not clear just how this suggestion for practice impacts on the Clements and Merriman and Perkins, Schwartz, Simmons' prescriptions for a programming environment. It would depend upon how involved the researchers see the instructor. I suspect that flexibility with regard to instructor intervention is important in their schemes. Thus, I see a potential problem given the Webb and Lewis results.

In conclusion, three of the four papers reviewed suggest extra-programming interventional environments. Fay and Mayer's research points to the need to be sensitive to the cognitive baggage students carry into these environments. Their work suggests a diagnosis and matching environmental component that would help to tailor the environment for maximum effectiveness regardless of whether the environment is structured for learning just programming skills, cognitive skills or both.

Together, these four papers offer significant contributions to thinking about and actually constructing appropriate learning environments around specific programming language instruction. I see the possibility of an important expansion of the much maligned notion of "computer literacy" and I appreciate

these researchers for their clear thinking and creative research.

Finally, I want to alert the research community to the impending invasion of a computer programming paradigm quite distinct from the currently popular procedural languages like BASIC, Logo, and Pascal. Prolog, is one of a class of logic programming languages that is declarative rather than imperative (i.e., procedural). One describes to the computer what the problem is rather than how to go about solving the problem. European researchers (Ennals and Briggs, 1985) are already claiming that Prolog is efficacious in teaching logic, problem solving and higher level skills.

I believe that North American researchers would do well to closely examine this new programming paradigm and the possible affects it might have on the kinds of environments being proposed here today.

REFERENCES

Ennals, R. and Briggs, J. (1985). Fifth Generation Computing: Introducing Micro-Prolog into the Classroom. Jrnl. of Educational Computing Research, 1, 97-111.

Seidman, R. H. (1981). The Effects of Learning a Computer Programming Language on the Logical Reasoning of School Children. Paper presented at the American Educational Research Association Annual Meeting, Los Angeles, CA. ERIC: ED 205 206.

Seidman, R. H. (1986). Transductive Reasoning and the Teaching of Conditional Logic to Children. Research Report 86-2, Occassional Paper Series. New Hampshire College Graduate School, Manchester, NH.